دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

کارشناسی ارشد علوم کامپیوتر گرایش داده کاوی

پروژه شماره چهار درس داده کاوی

نگارش

حدیث حق‌شناس جزی

استاد راهنما

مهدی قطعی

استاد مشاور

بهنام یوسفی مهر

آبان ۱۴۰۱

چکیده

در این گزارش هدف این است که با استفاده از دیتای مایکروسافت که شامل داده های GPS از ۱۸۲ نفر در طول ۵ سال می‌باشد به بررسی و مقایسه ۲ روش کاوش الگوهای پرتکرار بپردازیم. سپس این دو الگوریتم که ۲ شاخه اصلی شناخت الگوهای پرتکرار را برای ما انجام می‌دهند، با نام های Apriori و FP-Growth را بر روی دیتاست پیاده سازی کرده و میزان نتیجه دهی هر کدام را نسبت به مینیمم ساپورت های متفاوت نشان دهیم.

فصل اول

مقدمه

مقدمه

به منظور مقایسه روش های کاوش الگوهای تکراری Apriori و FP-Growth ، ابتدا به پیش پردازش و بررسی دیتاست می‌پردازیم. این بررسی شامل تبدیل دیتا از فایل های plt به یک فایل csv از طریق pickle میباشد و قسمت پیش پردازش نیز شامل بررسی ستون ها و تفکیک کردن ستون زمان (شامل تاریخ و ساعت دقیق) به تفکیک روز، رند کردن اعداد مختصاتی به ۳ رقم اعشار و حذف داده های با فراوانی پایین تر و ... می‌باشد. نهایتا با کمک کتابخانه های متفاوت پایتون و اعمال الگوریتم های کاوش و با تغییر مینیمم ساپورت سعی شده است که به یک نتیجه گیری کلی در زمینه مقایسه دو الگوریتم کاوش دست پیدا کنیم.

فصل دوم

پیش پردازش داده ها

۲-۱ معرفی دیتاست

دیتاست مورد بررسی در این پروژه در شرکت مایکروسافت از طریق مرکز تحقیقاتی آسیا جمع آوری شده است، که شامل داده های GPS در کشور از ۱۸۲ نفر در طول سال های ۲۰۰۷ تا ۲۰۱۲ می‌باشد. در این دیتاست که اطلاعات زمان (روز، ساعت، دقیقه، ثانیه) و مختصات طول و عرض و ارتفاع جغرافیایی میباشد غالبا در هر ۵ الی ۱۰ متر و یا در هر ۱ الی ۵ ثانیه ضبط شده است.

به طور کلی و پس از تبدیل دیتاست به فایل CSV شاهد حدود ۲۴ میلیون سطر خواهیم بود که هر سطر نمایانگر یک نقطه بر روی نقشه جغرافیایی میباشد.

۲-۲ پیش پردازش دیتاست

در ابتدا فایل دیتا را به CSV تبدیل میکنیم. اطلاعات کلی دیتاست در تصویر زیر قابل مشاهده است:

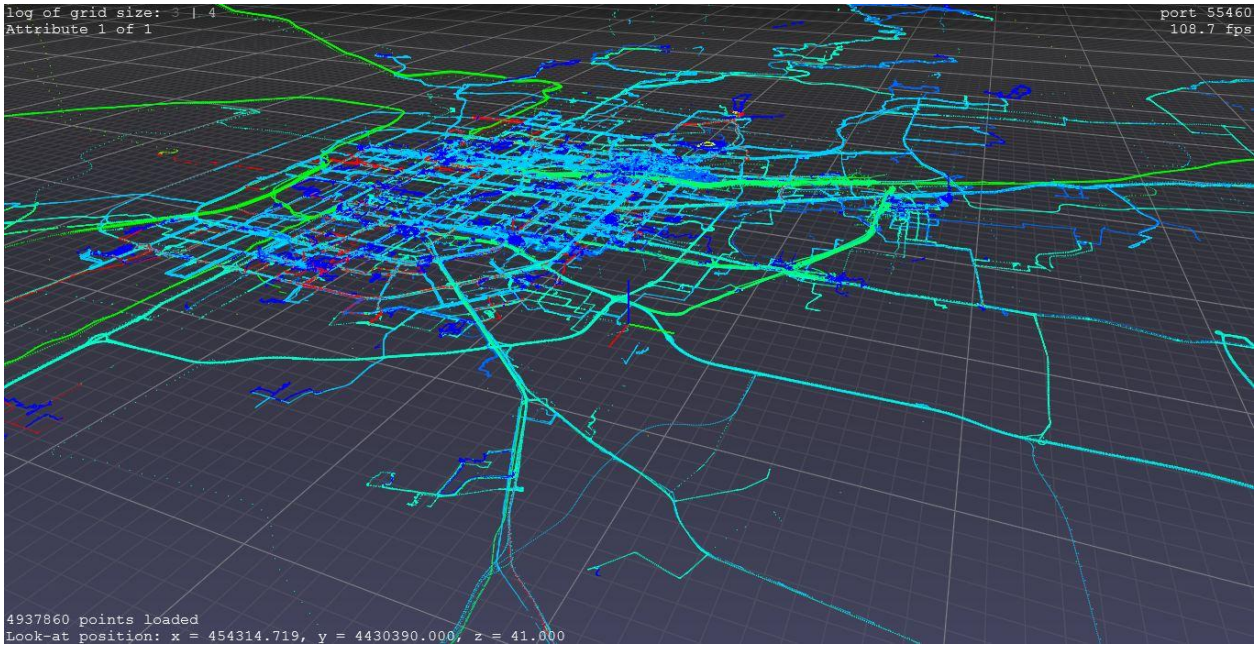
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24876978 entries, 0 to 21
Data columns (total 6 columns):
 #   Column  Dtype
---  ------  -----
 0   time    datetime64[ns]
 1   lat     float64
 2   lon     float64
 3   alt     float64
 4   label   int64
 5   user    int64
dtypes: datetime64[ns](1), float64(3), int64(2)
memory usage: 1.3 GB
```

نمایه کلی دیتاست به شکل زیر است:

	time	lat	lon	alt	label	user
0	2008-10-23 02:53:04	39.984702	116.318417	492.000000	0	0
1	2008-10-23 02:53:10	39.984683	116.318450	492.000000	0	0
2	2008-10-23 02:53:15	39.984686	116.318417	492.000000	0	0
3	2008-10-23 02:53:20	39.984688	116.318385	492.000000	0	0
4	2008-10-23 02:53:25	39.984655	116.318263	492.000000	0	0
...	...	...	...	...	...	...
17	2008-03-14 03:39:56	40.914867	111.710500	3802.493438	0	181
18	2008-03-14 03:41:17	40.914267	111.710333	3795.931759	0	181
19	2008-03-14 03:43:02	40.912467	111.710667	3795.931759	0	181
20	2008-03-14 03:43:28	40.911517	111.711317	3779.527559	0	181
21	2008-03-14 03:43:40	40.910933	111.711617	3802.493438	0	181

24876978 rows × 6 columns

در قسمت بعدی با استفاده از کتابخانه pyproj به بصری سازی داده بر روی جدول مختصات میپردازیم. رنگ های متفاوت در این نقشه نمایانگر لیبل های متفاوت میباشد (موجود در تراجکتوری بعضی اشخاص) که گویای مدل وسیله نقلیه (قطار، اتوبوس، ماشین و یا پیاده) است:



در این دیتاست داده های به صورت نقاط بسیار پراکنده و از دور مسیر های اصلی شهری را نمایش میدهند. به این دلیل که هدف ما پیدا کردن الگوریتم های پرتکرار است این نقاط را که با دقت ۶ رقم اعشار ضبط و ثبت شده اند به ۳ رقم اعشار تقلیل میدهیم.

	time	lat	lon	alt	label	user
0	2008-10-23 02:53:04	39.985	116.318	492.000	0	0
1	2008-10-23 02:53:10	39.985	116.318	492.000	0	0
2	2008-10-23 02:53:15	39.985	116.318	492.000	0	0
3	2008-10-23 02:53:20	39.985	116.318	492.000	0	0
4	2008-10-23 02:53:25	39.985	116.318	492.000	0	0
...	...	...	...	...	...	...
17	2008-03-14 03:39:56	40.915	111.710	3802.493	0	181
18	2008-03-14 03:41:17	40.914	111.710	3795.932	0	181
19	2008-03-14 03:43:02	40.912	111.711	3795.932	0	181
20	2008-03-14 03:43:28	40.912	111.711	3779.528	0	181
21	2008-03-14 03:43:40	40.911	111.712	3802.493	0	181

24876978 rows × 6 columns

حالا با توجه به ستون time که شامل اطلاعات زمانی به ترتیب : سال/ ماه / روز / ساعت / دقیقه / ثانیه میباشد بهتر است که برای پردازش بهتر و اینکه جزئیات این اطلاعات به این دقت مورد استفاده ما نیست، اطلاعات این ستون را به تفضیل روز در یک ستون مجزا (Date_D) بیاوریم:

```
1 print(df.head())
```

```
                 time        lat         lon    alt  label  user     Date_D
0 2008-10-23 02:53:04  39.984702  116.318417  492.0      0     0 2008-10-23
1 2008-10-23 02:53:10  39.984683  116.318450  492.0      0     0 2008-10-23
2 2008-10-23 02:53:15  39.984686  116.318417  492.0      0     0 2008-10-23
3 2008-10-23 02:53:20  39.984688  116.318385  492.0      0     0 2008-10-23
4 2008-10-23 02:53:25  39.984655  116.318263  492.0      0     0 2008-10-23
```

مرحله بعدی تبدیل طول و عرض و ارتفاع های داده شده به مختصات میباشد. با این وجود که ما از این موضوع اطلاع داشته ایم که این دیتاست شامل مختصات نقاط میباشند اما به تفکیک در ستون های مجزا آورده شده است لذا نیاز داریم که با تبدیل آنها به مختصات و به اضافه کردن ستون جدید بپردازیم (ارتفاع نقاط به دلیل

پستی بلندی زمین است و تاثیری روی مسیر ها ندارد به همین دلیل در این مرحله ارتفاع نقاط را در نظر نخواهیم گرفت) :

```
df['geo'] = df['lat'].astype(str) + '/' + df['lon'].astype(str)
```

```
print(df.head())
```

```
                 time        lat         lon    alt  label  user     Date_D  \
0 2008-10-23 02:53:04  39.984702  116.318417  492.0      0     0 2008-10-23
1 2008-10-23 02:53:10  39.984683  116.318450  492.0      0     0 2008-10-23
2 2008-10-23 02:53:15  39.984686  116.318417  492.0      0     0 2008-10-23
3 2008-10-23 02:53:20  39.984688  116.318385  492.0      0     0 2008-10-23
4 2008-10-23 02:53:25  39.984655  116.318263  492.0      0     0 2008-10-23

                   geo
0  39.984702/116.318417
1   39.984683/116.31845
2  39.984686/116.318417
3  39.984688/116.318385
4  39.984655/116.318263
```

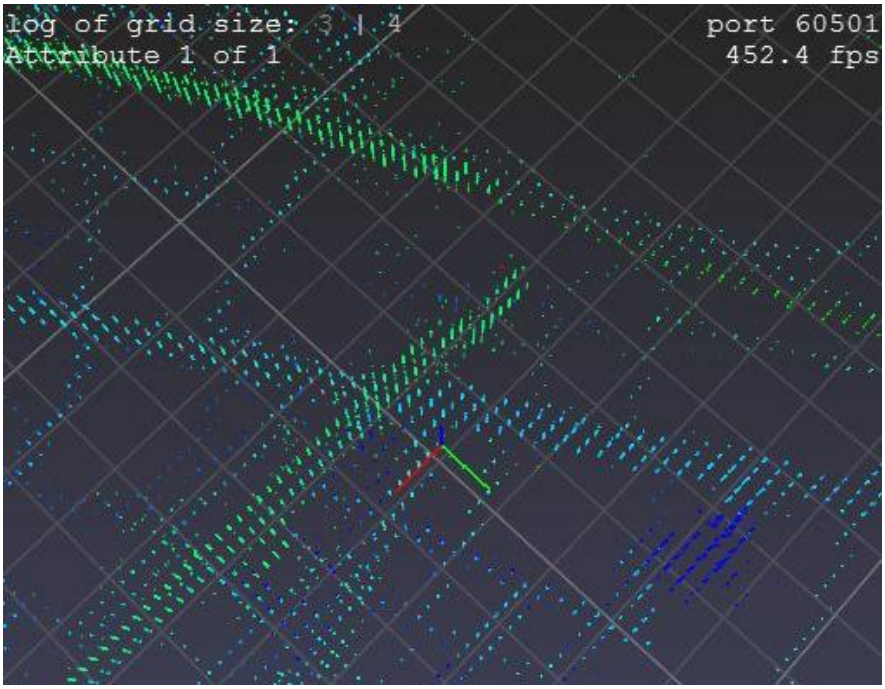
اکنون دیتاست را به تفضیل روز به صورت زیر داریم:

```
df = df.groupby('Date_D')['geo'].apply(set)
```

```
print(df)
```

```
Date_D
2000-01-02    {39.990964/116.327041, 39.993207/116.326827, 3...
2007-04-12    {39.98405/116.327016666667, 39.9490833333333/1...
2007-04-13    {39.9729166666667/116.312016666667, 39.9859/11...
2007-04-14    {40.0476166666667/116.181266666667, 40.0431/11...
2007-04-15    {40.1777333333333/116.232883333333, 40.2339833...
                                  ...
2012-07-23    {39.993026551/116.441454565, 40.031643026/116....
2012-07-24    {39.987286061/116.450479456, 39.985834677/116....
2012-07-25    {39.981012325/116.303367057, 39.979676548/116....
2012-07-26    {39.985295156/116.337938983, 39.992291774/116....
2012-07-27    {39.989751573/116.443712811, 39.992241512/116....
Name: geo, Length: 1878, dtype: object
```

پیش از رسیدگی به بخش بعدی یک بار دیگر به کمک کتابخانه pyproj دیتاست را رسم میکنیم. و میبینیم که نقاط تراکم بیشتری دارند و به دلیل حذف ۳ رقم اعشار بر روی هم افتاده اند:



فصل سوم

پیاده سازی و مقایسه الگوریتم ها

۳-۱ الگوریتم Apriori

در زیر به ۳ مینیمم ساپورت متفاوت این الگوریتم پیاده سازی شده است:

```
from PAMI.frequentPattern.basic import Apriori  as alg
db_geo = ("hadis.csv")
seperator=','
```

```
minimumSupportCount=2000
obj = alg.Apriori(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using Apriori algorithm
Total number of Frequent Patterns: 0
Total Memory in USS: 1119580160
Total Memory in RSS 1144082432
Total ExecutionTime in ms: 185.8395640850067
```

```
minimumSupportCount=900
obj = alg.Apriori(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using Apriori algorithm
Total number of Frequent Patterns: 8
Total Memory in USS: 1109757952
Total Memory in RSS 1134252032
Total ExecutionTime in ms: 178.2514295578003
```

```
minimumSupportCount=800
obj = alg.Apriori(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using Apriori algorithm
Total number of Frequent Patterns: 1059
Total Memory in USS: 1110724608
Total Memory in RSS 1135218688
Total ExecutionTime in ms: 183.05304956436157
```

همانطور که مشاهده میشود با کاهش مینیمم ساپورت از ۲۰۰۰ به ۹۰۰ در ابتدا زمان الگوریتم کاهش داشته است اما با کم کردن مجدد آن به ۸۰۰ دوباره افزایش زمان را شاهد هستیم. میتوان گفت که این الگوریتم زمان پردازش نسبتا طولانی دارد اما بهتر است در ادامه الگوریتم بعدی را نیز بررسی کنیم و سپس به مقایسه بپردازیم.

۳-۲ الگوریتم FP-Growth

در ابتدا با مینیمم ساپورت ۹۰۰ این الگوریتم را امتحان کرده و سپس مینمم ساپورت را پایین تر می آوریم:

```
from PAMI.frequentPattern.basic import FPGrowth  as alg
db_geo = ("hadis.csv")
minimumSupportCount=900  #Users can also specify this constraint between 0 to 1.
seperator=','
```

```
obj = alg.FPGrowth(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth algorithm
Total number of Frequent Patterns: 8
Total Memory in USS: 4949680128
Total Memory in RSS 4960882688
Total ExecutionTime in ms: 1.6268799304962158
```

```
from PAMI.frequentPattern.basic import FPGrowth  as alg
db_geo = ("hadis.csv")
minimumSupportCount=800  #Users can also specify this constraint between 0 to 1.
seperator=','
```

```
obj = alg.FPGrowth(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth algorithm
Total number of Frequent Patterns: 1059
Total Memory in USS: 4950024192
Total Memory in RSS 4961226752
Total ExecutionTime in ms: 3.354097843170166
```

```
minimumSupportCount=770
obj = alg.FPGrowth(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth algorithm
Total number of Frequent Patterns: 11123
Total Memory in USS: 1904967680
Total Memory in RSS 1928757248
Total ExecutionTime in ms: 3.7342867851257324
```

```
minimumSupportCount=740
obj = alg.FPGrowth(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth algorithm
Total number of Frequent Patterns: 114094
Total Memory in USS: 1907683328
Total Memory in RSS 1931472896
Total ExecutionTime in ms: 6.54522967338562
```

```
minimumSupportCount=710
obj = alg.FPGrowth(iFile=db_geo, minSup=minimumSupportCount, sep=seperator)    #initialize
obj.startMine()            #Start the mining process
obj.printResults()
```

```
Frequent patterns were generated successfully using frequentPatternGrowth algorithm
Total number of Frequent Patterns: 974864
Total Memory in USS: 2111102976
Total Memory in RSS 2134892544
Total ExecutionTime in ms: 24.95468258857727
```

همانطور که مشاهده میشود در مینیمم ساپورت ۹۰۰ زمان پردازش این الگوریتم ۱,۶ ثانیه بوده است و با پایین آوردن مقدار مینیمم ساپورت تا حدود ۷۱۰ این زمان تا حدود ۲۴ ثانیه (از مینیمم ساپورت ۷۴۰ به ۷۱۰ با شیب زیاد) زیاد شده است. این الگوریتم را میتوان ادامه داد که البته به میزان حافظه رم بستگی دارد و لذا بنده در این نقطه الگوریتم را متوقف میکنم.

جمع بندی

در این گزارش به بررسی دیتاست و سپس پیش پردازش آن و پیاده سازی دو الگوریتم کاوش الگو پرداختیم. همانطور که در تصاویر بالا مشاهده در این مقایسه این ۲ الگوریتم، با اختلاف الگوریتم Apriori زمان پردازش زیادتری دارد و میزان بهبود آن در مینیمم ساپورت های متفاوت تغییر چندانی نمیکند. اما در عوض الگوریتم FP-Growth بسیار سریع تر عمل میکند اما با تغییر جزئی تر مینیمم ساپورت تفاوت زمان پردازش آشکار تر است. همانطور که مشاهده کردیم در مینیمم ساپورت ۹۰۰، الگوریتم اول زمانی حدود ۱۸۵ ثانیه و الگوریتم دوم حدودا ۱,۶ ثانیه به طول انجامید که نشان از سریع تر بودن و برتری نسبتی الگوریتم FP-Growth دارد.

منابع و مراجع:

- سایت https://heremaps.github.io/pptk/tutorials/viewer/geolife.html
- استفاده از گزارش های جناب آقایان رافعی و شریفی
- کمک گرفتن از جناب آقای شریفی در قسمت راهنمایی های کلی برای دستورات

پایان